UNIVERSIDADE
DE VIGO

ESCOLA SUPERIOR DE ENXEÑARÍA

Memoria do Traballo de Fin de Grao que presenta

D. Julián Jiménez González

para a obtención do Título de Graduado en Enxeñaría Informática

SuperCars: Desarrollo de videojuego para dispositivos móviles

Xaneiro, 2015

Traballo de Fin de Grao Nº: EI 14/15 - 010

Titor/a: Alma María Gómez Rodríguez
Área de coñecemento: Linguaxes e Sistemas Informáticos
Departamento: Informática

Índice

1. Introducción

1.1. Descripción del sistema

SuperCars es un videojuego de carreras en 2D. En él, el jugador debe mover el vehículo a izquierda y derecha para evitar los obstáculos que van apareciendo en la carretera, así como adelantar a los oponentes. Se pretende que la aleatoriedad en la generación de obstáculos y oponentes provoque situaciones distintas en cada partida, así como el incremento de dificultad.

2. Análisis

2.1. Especificación de Requisitos Funcionales

En este apartado se reflejan todos los Requisitos Funcionales que se han establecido al inicio del proyecto o bien se han añadido posteriormente.
Éstos reflejan toda la funcionalidad que es imprescindible para considerar el proyecto como finalizado.

La nomenclatura es RF (Requisito Funcional) seguido de un número de identificación único. Ésta nomenclatura puede ser útil para relacionar cualquier parte de la documentación con los requisitos.

# Requisito	Descripción
RF01	Deben poder configurarse varios parámetros de la carrera como vueltas, rivales y dificultad
RF02	El jugador debe poder controlar su vehículo con unos botones que se muestren en pantalla
RF03	La vista del juego debe ser aérea (en 2D)
RF04	Debe haber un *HUD*[1] que muestre al jugador información sobre la carrera, como vueltas o posición
RF05	Debe guardarse un *ranking* de vueltas rápidas
RF06	La dificultad debe influír en el comportamiento de los rivales
RF07	El jugador debe poder pausar la carrera en cualquier momento

[1] *HUD son las siglas en inglés de Head-Up Display. Es una pantalla transparente que presenta la información al usuario sin que éste tenga que cambiar su punto de vista para verla. El origen proviene de que el usuario pueda ver la información con la cabeza erguida (Head-Up) y mirando al frente, sin necesidad de bajarla.*

2.2. Especificación de Requisitos No Funcionales

Aquí se detallan los Requisitos No Funcionales. Éstos reflejan características del sistema que, si bien su incumplimiento no hace que el sistema deje de funcionar, sí pueden influír en la experiencia del usuario al utilizar la aplicación.

La nomenclatura sigue el mismo patrón que en los *Requisitos Funcionales (2.1)*, siendo RNF (Requisitos No Funcionales) seguido de un número de identificación único, además de poseer la misma utilidad.

# Requisito	Descripción
RNF01	La aplicación debe mantener una tasa de *frames per second* superior a 30
RNF02	La tasa de *frames per second* debe ser estable (no variar más de un 25% cada segundo
RNF03	La carrera debe poder contener 40 o más obstáculos simultáneos
RNF04	La aplicación debe funcionar en modo *portrait*[2] para ofrecer suficiente visibilidad al jugador
RNF05	Al iniciar la aplicación, el menú principal debe ser visible en menos de 5 segundos
RNF06	La transición entre escenas no debe durar más de 2 segundos

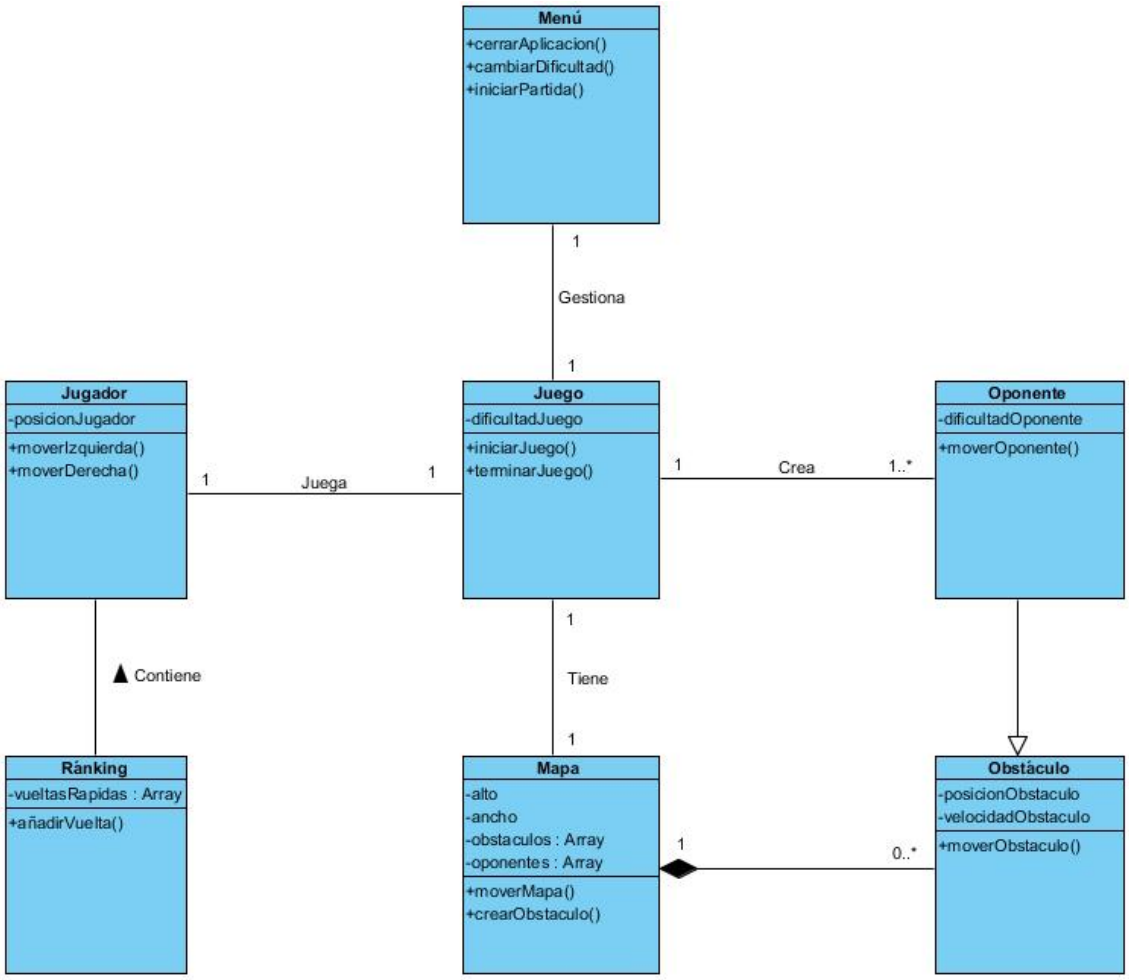
2.3. Modelo de Dominio

En el Modelo de Dominio se reflejan las clases (conceptuales) que se consideran significativas para el dominio del problema.

Estas clases no tienen por qué coincidir con las implementadas finalmente, ya que éstas últimas pueden variar en función del entorno de desarrollo e incluso del de despliegue.
La lógica y funcionalidades descritas en estas clases sí deben mantenerse.

La versión final del Modelo de Dominio que refleja el diseño real de la aplicación se puede encontrar en el apartado *3.1.8*.

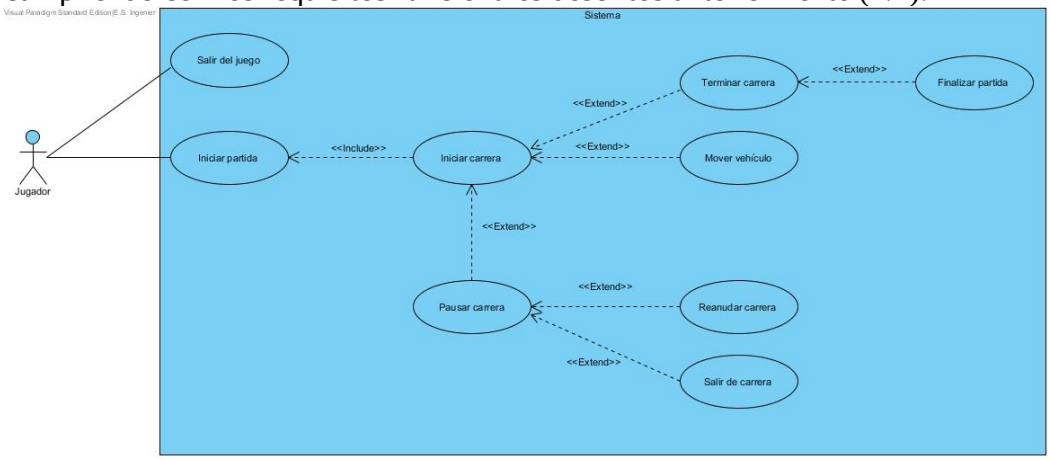
[2] *Portrait hace referencia a la orientación de la aplicación, que debe extender su lado más largo de arriba a abajo de la pantalla (orientación vertical)*



2.4. Descripción de Casos de Uso

2.4.1. Diagrama de Casos de Uso

Este diagrama representa las interacciones del usuario con el sistema, cumpliendo con los requisitos funcionales descritos anteriormente (*2.1*).



A continuación se realiza una descripción detallada de los Casos de Uso, que incluye:

- Breve descripción.

- Precondiciones.
- Postcondiciones.
- Flujo de eventos.
- Flujo alternativo (opcional).

Se obvia el actor debido a que en todos ellos es el jugador.

2.4.2. Descripción detallada de Casos de Uso

2.4.2.1. Iniciar partida

Iniciar partida			
Descripción breve	El jugador pulsa el botón *Play* en *MainMenu* y la escena cambia a *RaceConf*		
Precondiciones	La escena activa es *MainMenu*		
Postcondiciones	La escena activa es *RaceConf*		
Flujo Principal		**Jugador**	**Sistema**
	1	Pulsa el botón *Play*	
	2		Cambia escena activa de *MainMenu* a *RaceConf*

2.4.2.2. Iniciar carrera

Iniciar carrera			
Descripción breve	El jugador configura las opciones de carrera y ésta da comienzo de acuerdo con ellas		
Precondiciones	La escena activa es *RaceConf*		
Postcondiciones	La escena activa es *Race*		
Flujo Principal		**Jugador**	**Sistema**
	1		Muestra parámetros de configuración de la carrera
	2	Ajusta los parámetros de configuración	
	3		Recoge la configuración introducida e inicia la carrera de acuerdo a dicha configuración

2.4.2.3. Mover vehículo

Mover vehículo			
Descripción breve	El jugador pulsa un botón de dirección y el vehículo se mueve al sitio correspondiente		
Precondiciones	La escena activa es *Race*		
Postcondiciones	El vehículo se ha movido a la posición deseada		
Flujo Principal		**Jugador**	**Sistema**
	1	Pulsa botón de movimiento *leftArrow* o *rightArrow*	
	2		Identifica cuál de los

			botones ha sido pulsado
	3		Mueve el vehículo al lado correspondiente
	4		Se queda a la espera de una nueva interacción
Flujo Alternativo [A3]		**Jugador**	**Sistema**
	1		Si está a la izquierda y se ha pulsado *leftArrow*, no hace nada. Vuelve al paso 4
Flujo alternativo [A3]		**Jugador**	**Sistema**
	1		Si está a la derecha y se ha pulsado *rightArrow*, no hace nada. Vuelve al paso 4

2.4.2.4. Terminar carrera

Terminar carrera			
Descripción breve	El jugador alcanza la línea de meta de la última vuelta y la carrera finaliza		
Precondiciones	La escena activa es *Race*		
Postcondiciones	La escena activa es *EndRace*		
Flujo Principal		**Jugador**	**Sistema**
	1	Alcanza la línea de meta	
	2		Finaliza la carrera
	3		Cambia escena activa de *Race* a *EndRace*

2.4.2.5. Finalizar partida

Finalizar partida			
Descripción breve	El jugador vuelve a *MainMenu* pulsando *back* en la escena *EndRace*, una vez terminada la carrera		
Precondiciones	La escena activa es *EndRace*		
Postcondiciones	La escena activa es *MainMenu*		
Flujo Principal		**Jugador**	**Sistema**
	1	Pulsa *back*	
	2		Cambia escena activa de *EndRace* a *MainMenu*

2.4.2.6. Pausar carrera

Pausar carrera			
Descripción breve	El jugador pulsa el botón *menú* en la escena *Race* y se pausa el juego para mostrar *RaceMenu*		
Precondiciones	La escena activa es *Race*		
Postcondiciones	La escena activa es *RaceMenu*		
Flujo Principal		**Jugador**	**Sistema**
	1	Pulsa botón *menu*	

	2		Pausa escena *Race*
	3		Muestra escena *RaceMenu*

2.4.2.7. Reanudar carrera

Reanudar carrera		
Descripción breve	El jugador pulsa *resume* en *RaceMenu* y se reanuda la carrera	
Precondiciones	La escena activa es *RaceMenu*	
Postcondiciones	La escena activa es *Race*	
Flujo Principal	**Jugador**	**Sistema**
	1 Pulsa *resume*	
	3	Cambia escena activa de *RaceMenu* a *Race*

2.4.2.8. Salir de carrera

Salir de carrera		
Descripción breve	El jugador pulsa el botón *Exit* en *RaceMenu* y la escena cambia a *MainMenu*	
Precondiciones	La escena activa es *RaceMenu*	
Postcondiciones	La escena activa es *MainMenu*	
Flujo Principal	**Jugador**	**Sistema**
	1 Pulsa el botón *exit*	
	2	Finaliza la escena *Race*
	3	Cambia escena activa de *RaceMenu* a *MainMenu*

2.4.2.9. Salir del juego

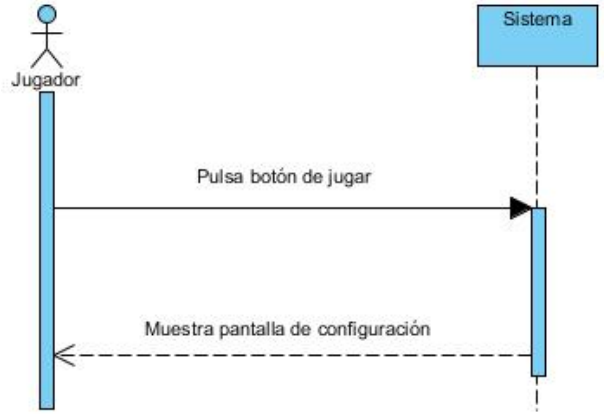
Salir del juego		
Descripción breve	El jugador pulsa el botón *exit* de *MainMenu* y la aplicación se cierra	
Precondiciones	La escena activa es *MainMenu*	
Postcondiciones	Aplicación cerrada correctamente	
Flujo Principal	**Jugador**	**Sistema**
	1 Pulsa *exit* en *MainMenu*	
	2	Cierra la aplicación

2.5. Diagramas de Secuencia

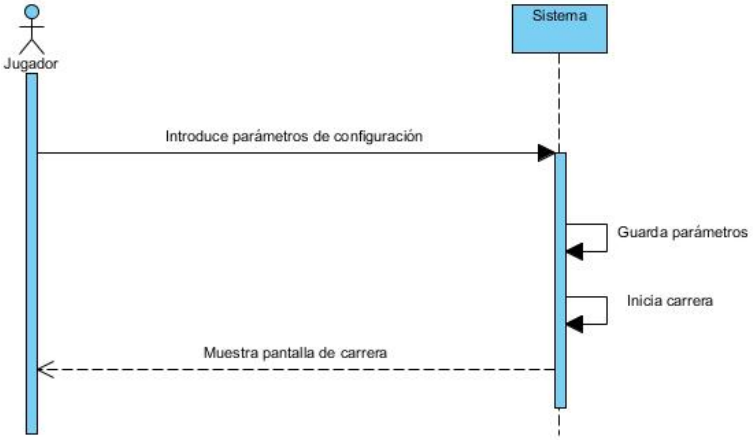
Aquí se muestran los diagramas de secuencia que reflejan el flujo de información que se produce entre el usuario y el sistema, según las clases y casos de uso de análisis.

Ésto es ampliado con los *Diagramas de Secuencia del Sistema (3.2)* que reflejan también el flujo interno del sistema.

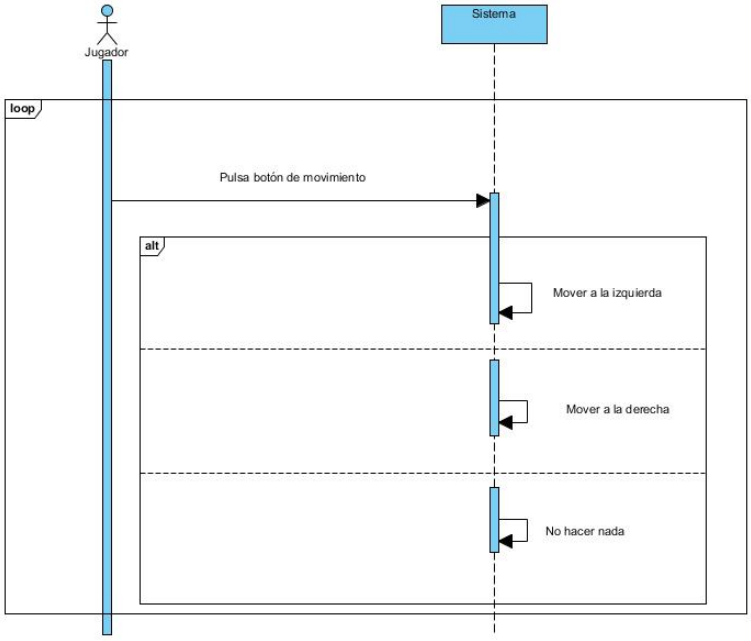
2.5.1. Iniciar partida



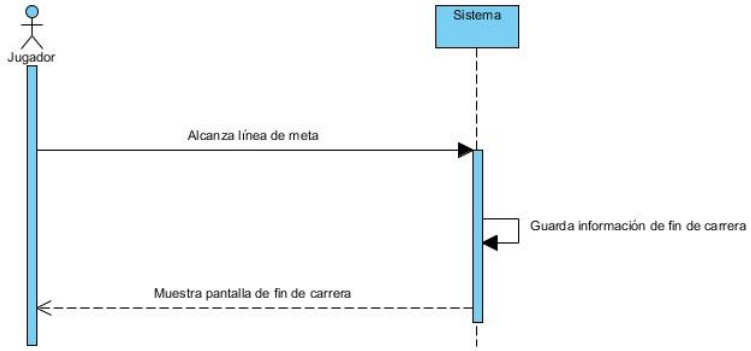
2.5.2. Iniciar carrera



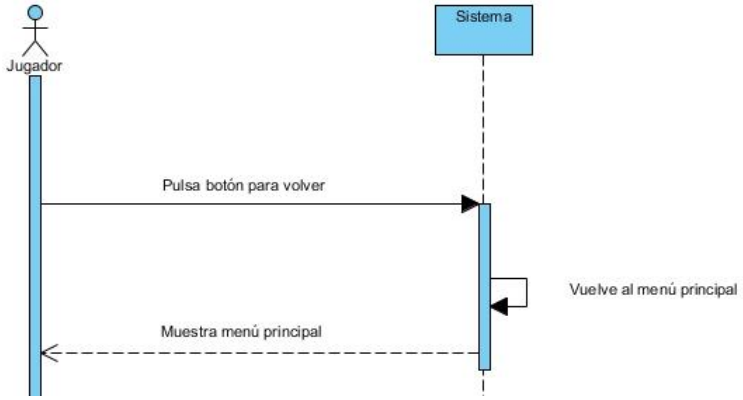
2.5.3. Mover vehículo



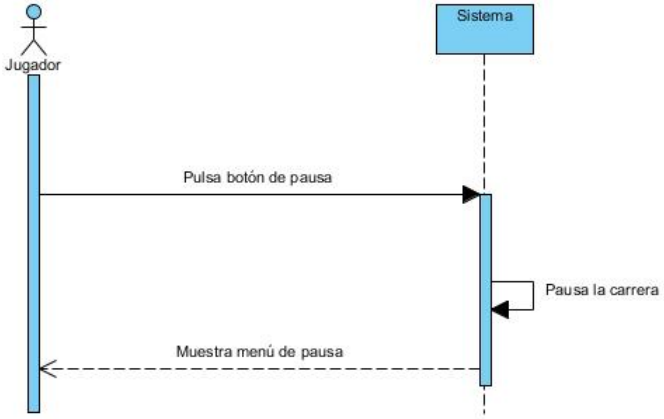
2.5.4. Terminar carrera



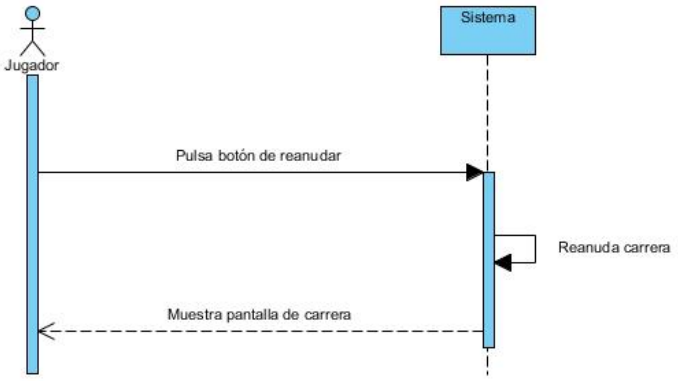
2.5.5. Finalizar partida



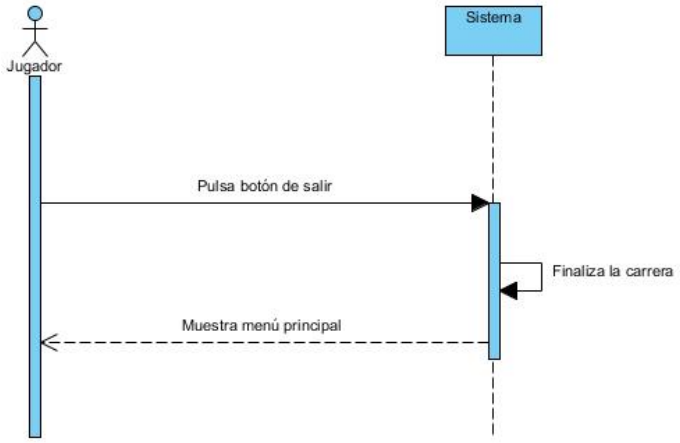
2.5.6. Pausar carrera



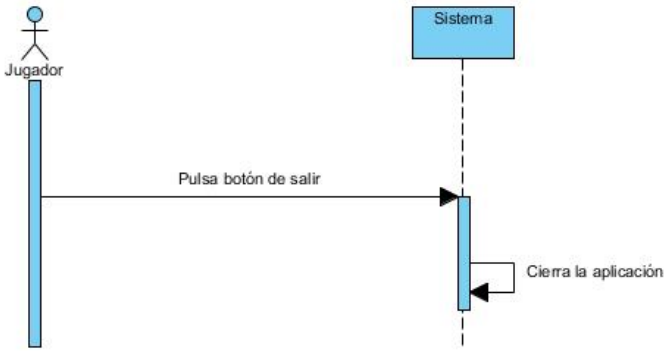
2.5.7. Reanudar carrera



2.5.8. Salir de carrera



2.5.9. Salir del juego



3. Diseño

Las variaciones en el diseño con respecto al *Análisis (2)* están producidas por la adecuación del desarrollo al *framework* Cocos2d-x.

A continuación se muestran las clases parciales finales, así como el diagrama de clases total.

3.1. Diagramas de clases parciales

En este apartado se detallan las clases parciales del sistema, que se pueden identificar con una escena del juego (exceptuando AppDelegate).
Todas las clases tienen su diagrama parcial, una breve explicación y sus métodos y variables más importantes.

Existen ciertas variables y métodos que son comunes a casi todas las clases, por lo que se detallan aquí:

- *origin*: variable de tipo *Vec2* que contiene las coordenadas X e Y del origen.
- *visibleSize*: variable de tipo *Size* que contiene la altura y el ancho de la pantalla.
- *createScene()*: es el constructor de la escena, por lo que devuelve un objeto escena básico.

- *init()*: función que se ejecuta siempre y contiene las llamadas principales de la clase. Se puede considerar como una función *main*[3], por lo que el resto de métodos son llamados (directa o indirectamente) desde éste.
- *createMenu()*: crea y coloca los botones del menú correspondiente a esa escena.

3.1.1. AppDelegate

Esta es una clase base propia de Cocos2d-x. Su función es la de gestionar el inicio y fin de la aplicación.

- *applicationDidFinishLaunching()*: se ejecuta cuando la aplicación termina de abrirse.
- *applicationDidEnterBackground()*: se ejecuta cuando la aplicación pasa a segundo plano.
- *applicationWillEnterForeground()*: se ejecuta cuando la aplicación vuelve activarse tras *applicationDidEnterBackground()*.

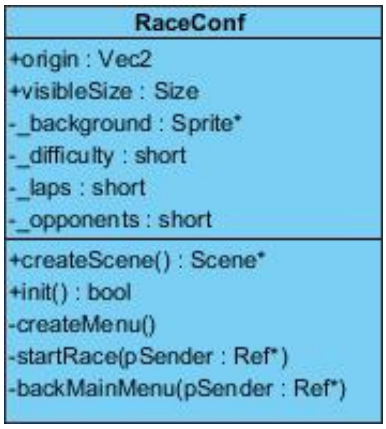
3.1.2. MainMenu

Esta clase se corresponde con el menú principal del juego. Es por tanto la escena creada desde *AppDelegate*.

- *playGame(Ref* pSender)*: crea una nueva instancia de *RaceConf* y reemplaza a ésta.
- *showRanking(Ref* pSender)*: crea una nueva instancia de *RankingMenu* y reemplaza a ésta.
- *exitGame(Ref* pSender)*: finaliza la aplicación.

[3] *Función típicamente llamada al arrancar un programa*

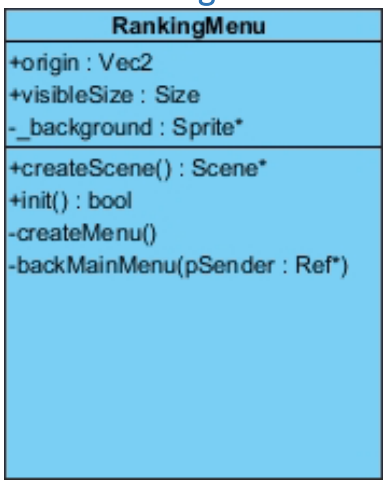
3.1.3. RaceConf



Esta clase corresponde a la escena de configuración de la carrera.

- *startRace(Ref* pSender)*: recoge y guarda los parámetros introducidos, después crea una nueva instancia de *RaceScene* y reemplaza a ésta.
- *backMainMenu(Ref* pSender)*: vuelve a *MainMenu*.

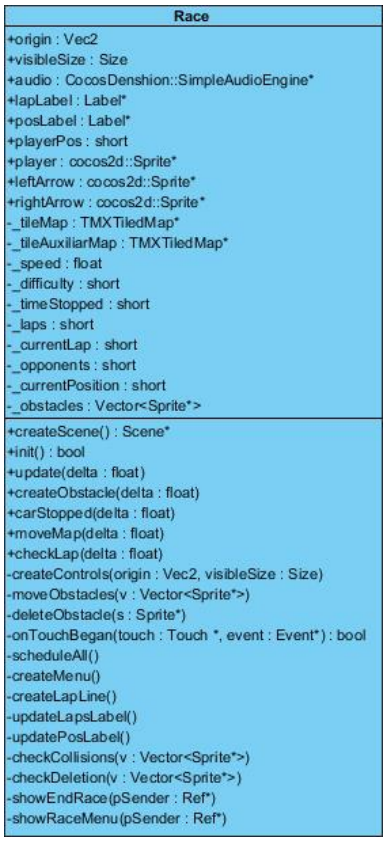
3.1.4. RankingMenu



Esta clase corresponde a la escena de visualización del *ranking* de vueltas rápidas.

- *backMainMenu(Ref* pSender)*: vuelve a *MainMenu*.

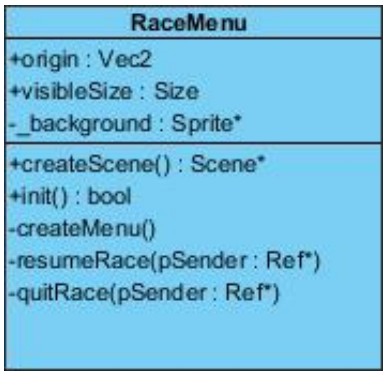
3.1.5. Race



Esta clase es la escena principal del juego. Contiene todo lo visible en la escena de carrera.

- *_difficulty*, *_laps* y *_opponents*: estas variables se recogen de la información introducida en *RaceConf*.
- *_speed*: esta variable controla cómo de rápido se mueve el mapa y los rivales. Se calcula a partir de *_difficulty* y *_opponents*.
- *scheduleAll()*: este método programa (*schedule)* los métodos que deben ejecutarse cada *frame* o cada tiempo estimado.
- *update(float delta)*: bucle principal del juego. Se ejecuta cada *frame* y *delta* es el tiempo transcurrido entre el *frame* actual y el anterior (este tiempo es variable).
  Desde este método se realizan llamadas a *updatePosLabel()*, *moveObstacles(_obstacles)*, *checkCollisions(_obstacles)* y *checkDeletion(_obstacles)*.
- *carStopped(float delta)*: éste método pasa a ser el bucle principal cuando el vehículo del jugador se para (ha colisionado con un obstáculo). Tras un tiempo llama de nuevo a *scheduleAll()* y se desactiva.
- *moveMap(float delta)*: mueve el mapa en función de *_speed*.
- *moveObstacles(Vector<Sprite*> v)*: mueve los obstáculos en función de *_speed* para generar la sensación de que éstos están fijos en el mapa.
- *createObstacle(float delta)*: crea un obstáculo cada *delta* segundos en una posición semi-aleatoria.
- *checkLap(float delta)*: comprueba cada *delta* segundos si la vuelta actual es igual al número máximo de vueltas, para terminar la carrera.

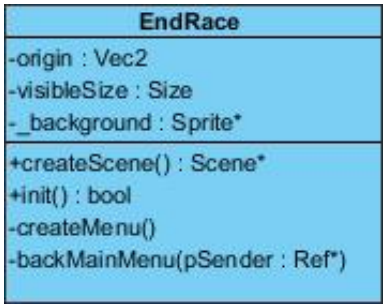
3.1.6. RaceMenu



Esta escena es el menú llamado desde *Race*.

- *resumeRace(Ref* pSender)*: reanuda la carrera.
- *quitRace(Ref* pSender)*: finaliza la carrera, crea una nueva instancia de *MainMenu* y se reemplaza por ésta.

3.1.7. EndRace

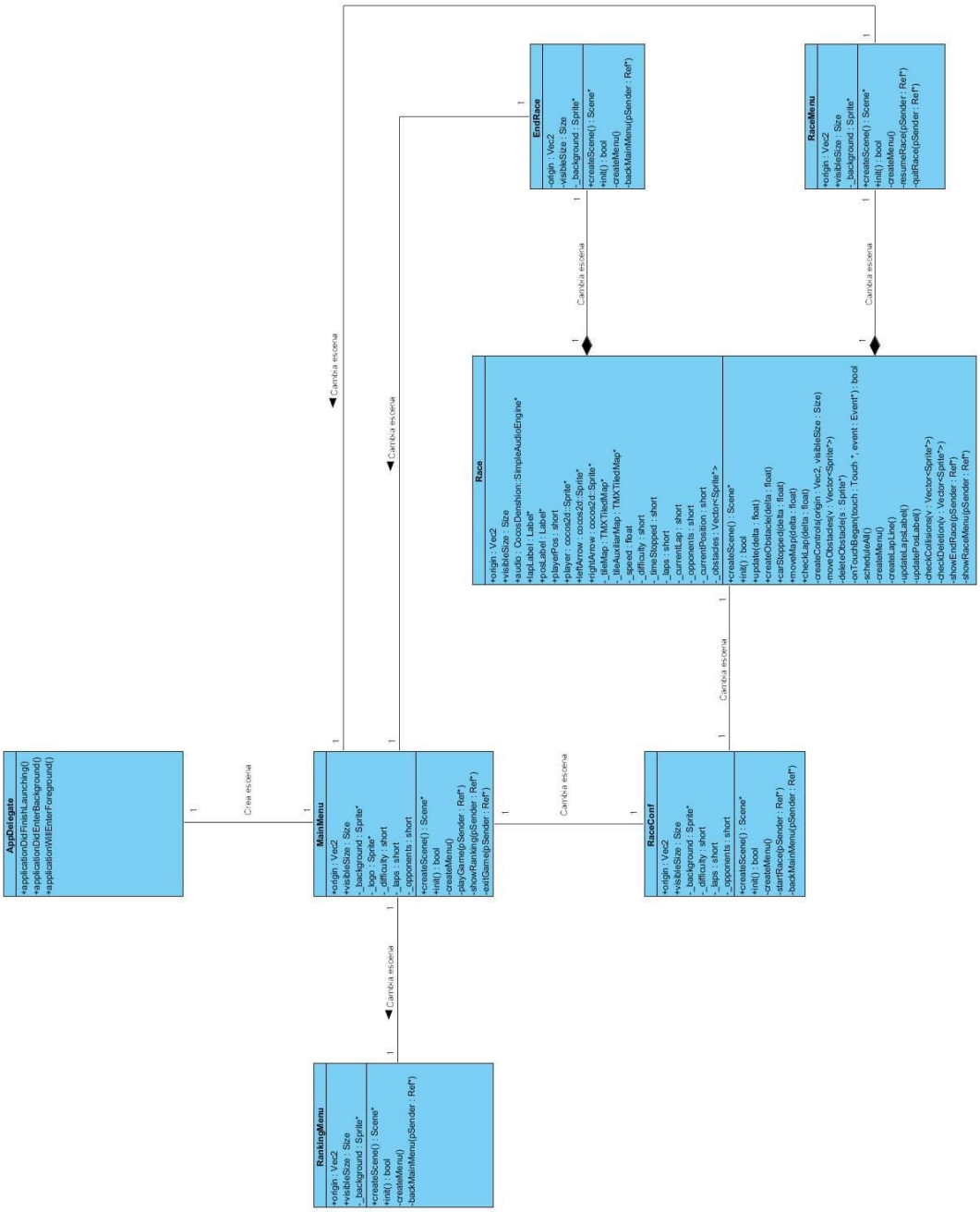


Esta es la escena que sigue a *Race* cuando la carrera termina. Muestra el tiempo obtenido y permite volver a *MainMenu*.

- *backMainMenu(Ref* pSender)*: devuelve a *MainMenu*.

3.1.8. Diagrama de clases total

Este es el diagrama de clases total que finalmente refleja el diseño de la aplicación.
Es diferente del Modelo de Dominio *(2.3)* de la fase de *Análisis* pero mantiene la funcionalidad descrita.



De las diferencias más importantes entre éste diagrama de clases final y el *Modelo de Dominio (2.3)* de la fase de *Análisis*, cabe destacar:

- La entidad Jugador no es necesario representarla como una clase separada, ya que finalmente es una variable de la escena *Race (3.1.5)* y la clase *Sprite* propia de Cocos2d-x es suficiente para implementar la funcionalidad deseada.
  Sucede lo mismo con las clases *Mapa*, *Obstáculo* y *Oponente*. En el caso de éstos últimos incluso son tratados en un mismo vector, diferenciados por un *Tag*[4]
-

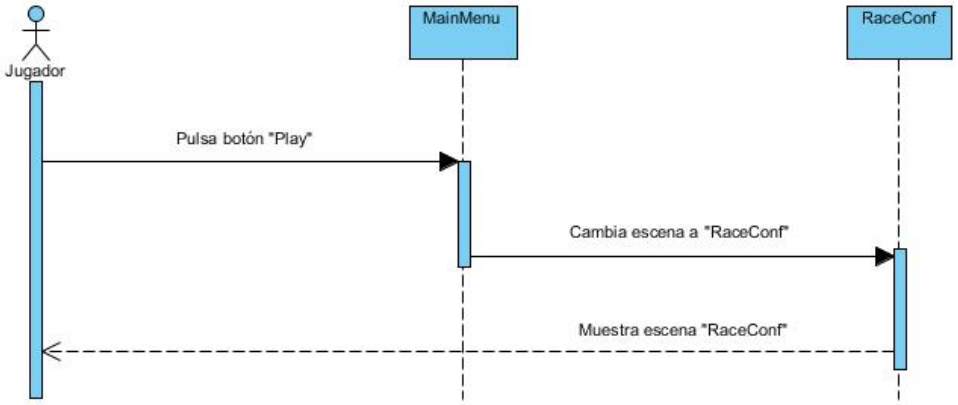
[4] *Cocos2d-x utiliza Tags en sus clases como un identificador que puede ser único o compartido por distintos elementos.*

3.2. Diagramas de Secuencia del Sistema

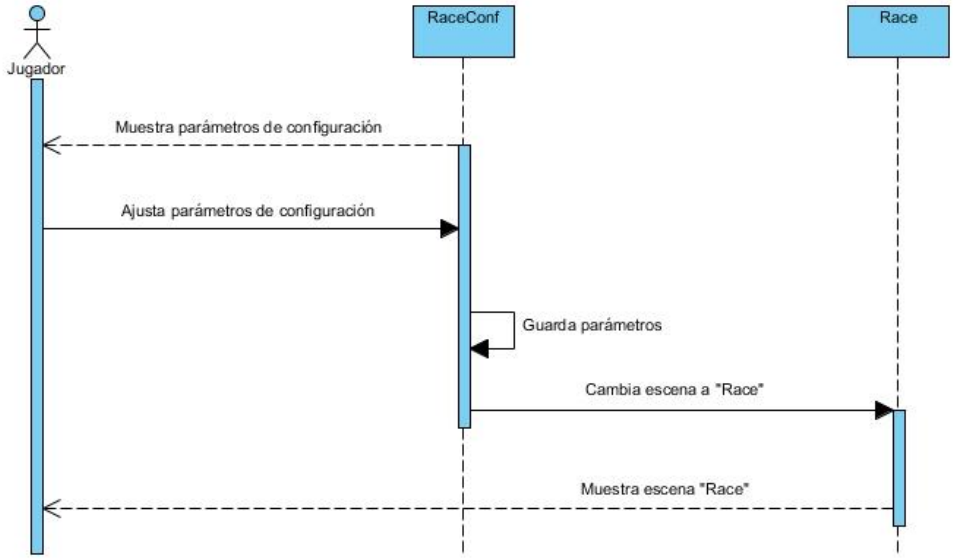
En estos diagramas se pueden ver las transiciones de información y los cambios que se producen en el sistema debido a la interacción del Jugador.

Por motivos de legibilidad se ha decidido describir las relaciones con frases cortas en lugar de indicar el nombre del método correspondiente, aunque cada relación entre clases del sistema corresponde con la llamada a un método.

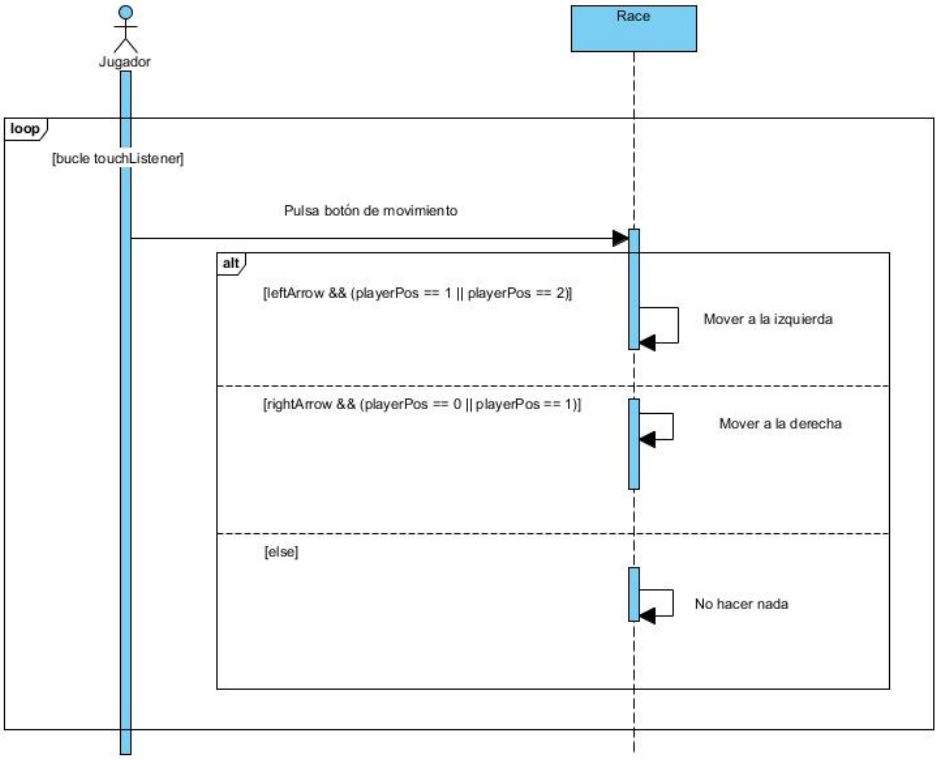
3.2.1. Iniciar partida



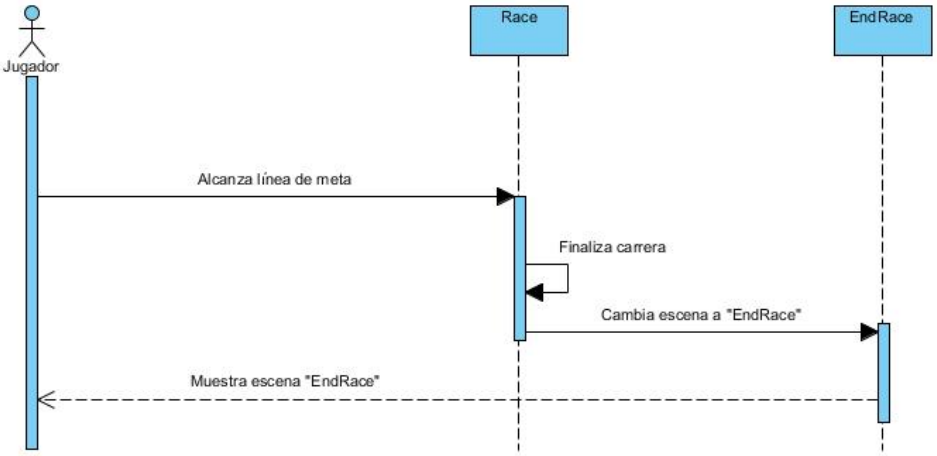
3.2.2. Iniciar carrera



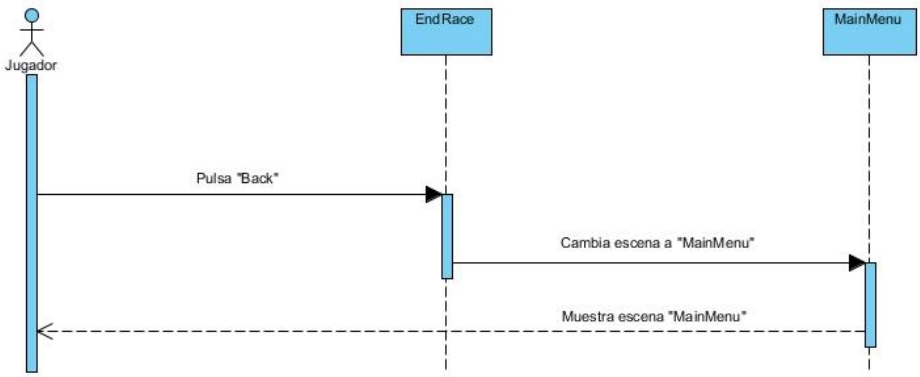
3.2.3. Mover vehículo



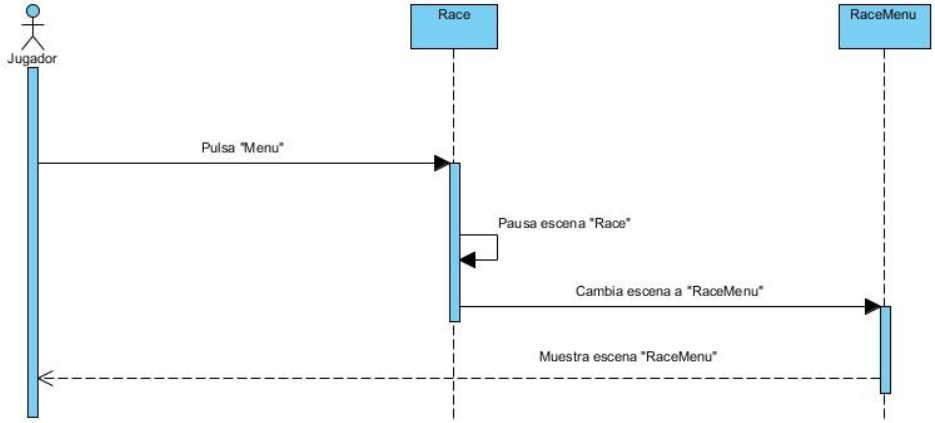
3.2.4. Terminar Carrera



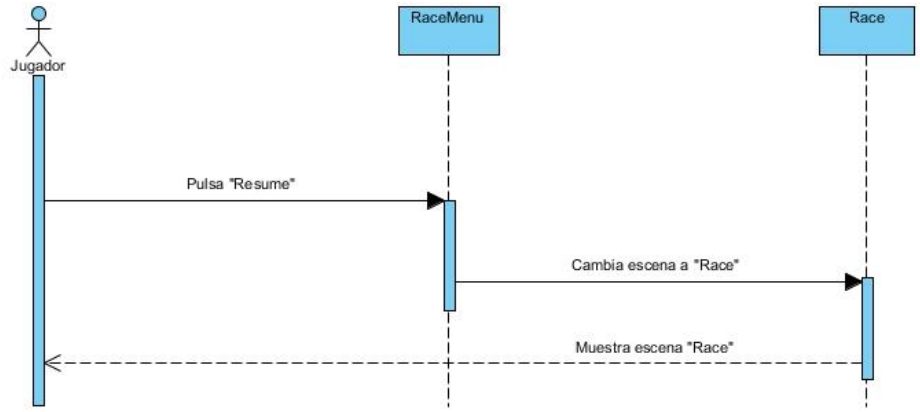
3.2.5. Finalizar partida



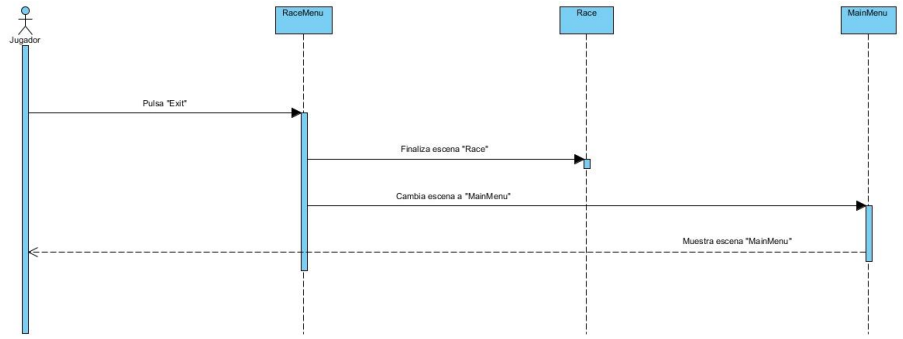
3.2.6. Pausar carrera



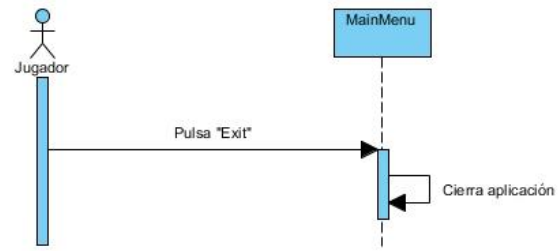
3.2.7. Reanudar carrera



3.2.8. Salir de carrera



3.2.9. Salir del juego



4. Detalles de implementación

Aquí se explican todos los detalles de implementación que se considera que son importantes bien por ser características muy específicas de Cocos2d-x o de este proyecto concreto.

4.1. Jugador

4.2. Mapa

4.3. Bucle principal

4.4. Métodos con ejecución programada

4.5. Colisiones

4.6. Eventos